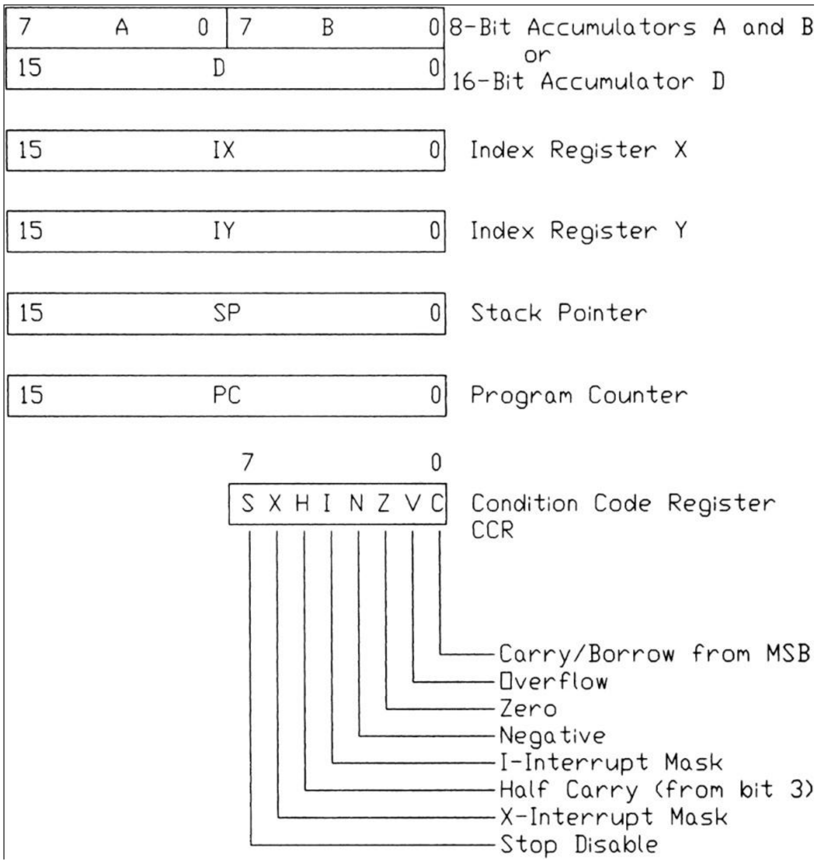
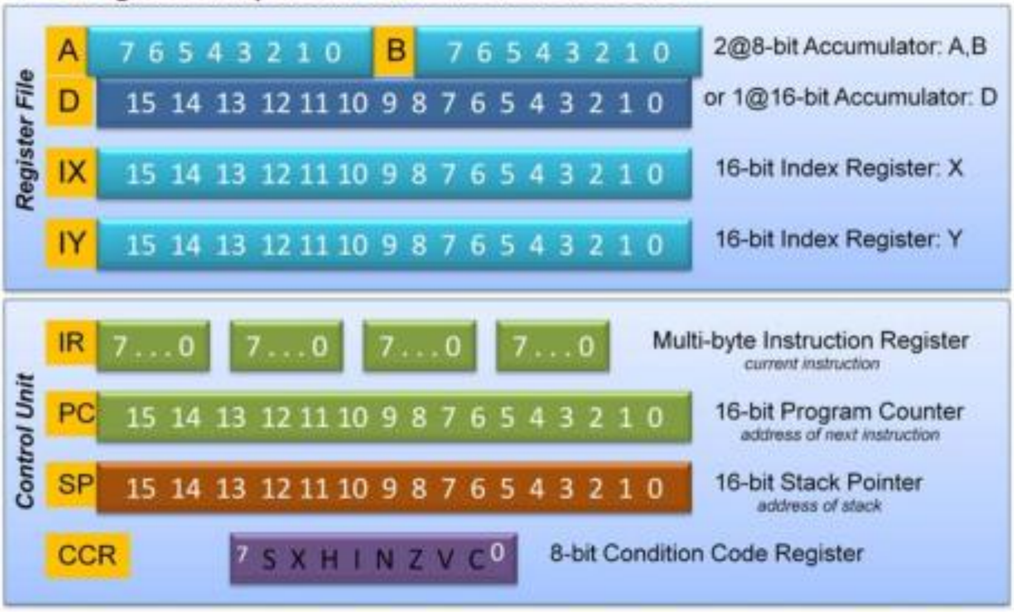
Ch (2)

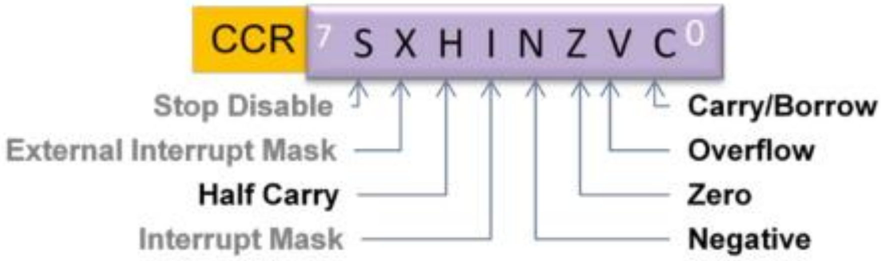
CPU Registers

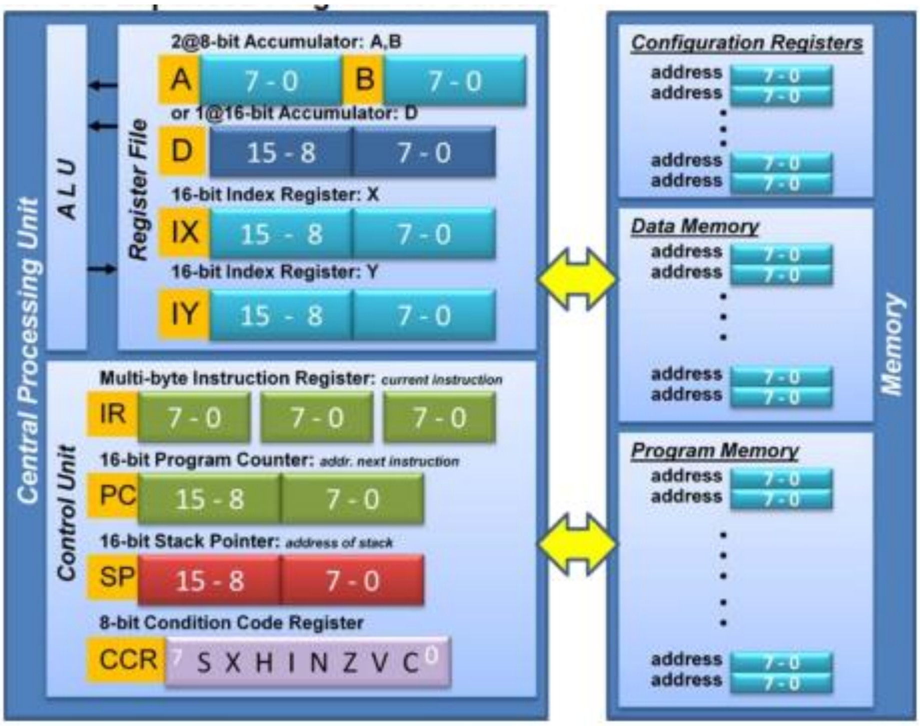


لتوضيح هذه الرسمه اكثر ،، الرسمتين الجايين للفهم
المخازن دي بتكون فين ...

CPU Registers important to instruction execution







Register File

- store data in/out of memory or ALU
- Accumulators: 2 @ 8-bit OR 1 @ 16-bit

*There are two 8-bit accumulators (ACCA and ACCB)

* Each may be a source or destination operand for 8-bit instructions.

اما بجيب بيانات من الميموري او من اطراف الدخل و عايز اعمل عليها عمليه حسابيه او منطقيه بحطها في احد المخازن التي تسمى

ACC A , ACC B

- Index Registers: 2 @ 16-bit

• Index registers X and Y: The two 16-bit index registers are used primarily for indexed addressing (will be studied in addressing modes). There are also some arithmetic instructions involving the index registers.

e.g. INX (increments the contents of index register X, i.e., (IX) <<< (IX)+$0001)

يُخزن فيه عناوين ،، ويجرى فيه عمليات حسابيه ،،، بحيث اني بقدر ازود عليه 1 مثلا بأمر ال INX ,INY

• Control Unit

- Program Counter (PC)

• holds address of next instruction, 16-bit

فاكرين في شبتر 1 فيه مرحله في تنفيذ اي برنامج اسمها الــ

Fetch

كنت بقول ان المعالج بيبدأ يتعامل مع سطر سطر من البرنامج يجلبه بعنوانه،، عنوان السطور دول بيتكتب في الــ

PC

- Stack Pointer (SP)

• holds address of stack (special block of memory), 16-bit

تعريف الـاستاك في شبتر 4

- Condition Code Register (CCR)

• holds "flag" values generated by last instruction executed, 8-bit

• known as Status Register in other controllers

هذا الجزء لفهم الــ

CCR , 8 bits

ALU Status Flags: each condition tested after each instr. exec.

- C: Carry Flag

•= 1 if carry (addition) or borrow (subtraction) occurs in instr. exec.

-V: Overflow Flag

= 1 if 2's complement overflow occurs in instr. exec.

-Z: Zero Flag: =1 if ALU result = 0

-N: Negative Flag: =1 if MSU of ALU result is 1 (i.e., negative in S2C)

- H: Half-Carry Flag: =1 if carry from lower nibble (nibble = ½ byte =4 bits)

CCR Examples :

Instructions & resulting CCR flag status

- 8-bit addition

B7 + 4A

في المثال بفرض ان ال

ACC A

عايز اعمل فيه عمليه الجمع بين الرقمين اللي فوق دول و اعرف الناتج هيغير ايه في

CCR

الارقام اللي فوق مكتوبه بنظام السداسي عشر هحوله لنظام ثنائي عشان اعرف اجمع

B7= 1011 0111

4A= 0100 1010

خذوا بالكم من اللي جاي لاني كنت قولته بطريقه تانيه في السكشن لكن الطريقه اللي هقولها هيا المستخدمه و الصحيحه ... ايه الطريقه بقى و بتقول ايه ؟
بتقول ان اي رقم بيدخل للمخزن بيكون معبر عنه بطريقه ال

2's complement

If MSB =1 >> number is negative

If MSB =0 >> number is positive

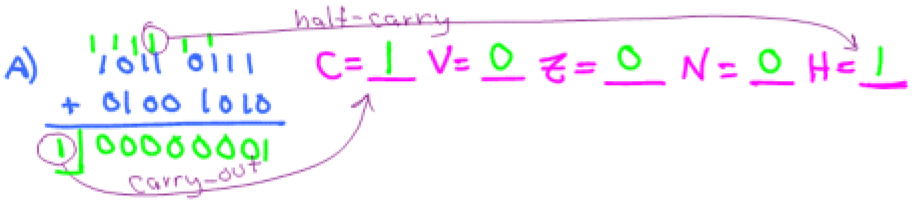
معنى كده ان الرقمين اللي فوق

B7 >>>negative

$=(1*2^0)+(1*2^1)+(1*2^2)+(0*2^3)+(1*2^4)+(1*2^5)+(0*2^6)-(1*2^7) = 1+2+4+16+32-128 = -73$

4A >>> positive

$=(0*2^0)+(1*2^1)+(0*2^2)+(1*2^3)+(0*2^4)+(0*2^5)+(1*2^6)-(0*2^7) = 2+8+64 = 74$



مفروض اما تجمع 73- و74 يطلع الناتج 1هلاقي اول 8 بت بيمثلوا ال1 و البت التاسعه دي اسمها

Discard carry_out

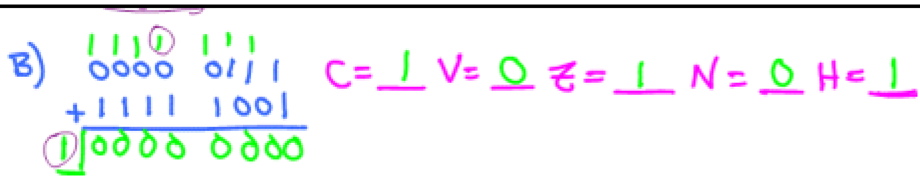
ده هيخلي ال

C (carry) in condition code register(CCR) = 1

تمام؟؟!

---

07 + F9



07 = 0000 0111 = + 7

F9 =1111 1001

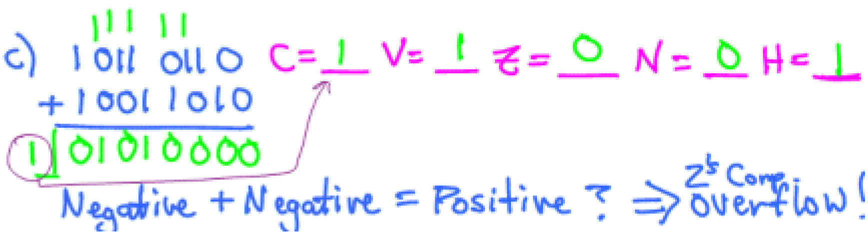
$=(1*2^0)+(0*2^1)+(0*2^2)+(1*2^3)+(1*2^4)+(1*2^5)+(1*2^6) - (1*2^7) = 1+0+0+8+16+32+64 - 128 = -7$

SO, 7 +(-7)=0

اذا البت التاسعه في ناتج الجمع هوا

Discard carry out , c=1 .

B6 + 9A



B6 = 1011 0110

$=(0*2^0)+(1*2^1)+(1*2^2)+(0*2^3)+(1*2^4)+(1*2^5)+(0*2^6) - (1*2^7)= 2+4+16+32-128= -74$

9A= 1001 1010

$=(0*2^0)+(1*2^1)+(0*2^2)+(1*2^3)+(1*2^4)+(0*2^5)+(0*2^6) - (1*2^7)= 2+8+16-128= -102$

-74 -102 = -176 (1 0101 0000)>> 9 bit !!

هلاقي ناتج الجمع محتاج فيه ال9بت كلهم ...يبقى كده حصل اوفر فلو ... و الكاري برده بـ1

ASM Instruction Format

صوره الامر المكتوب بالاسمبلي
مكون من اربع اجزاء
Label
Instraction mnemonic
Operand
Comment
يكتبوا بالترتيب الاتي :

-LABEL INSTR_MNEMONIC OPERAND(S) COMMENT

وظيفة كل منهم :

LABEL: identifies location of a line of code
- used by Assembler (compiler); not part of actual instructions
- generally used for looping (jump, branch)

• MNEMONIC: text code for each ASM instruction
- translated by Assembler to a specific instr. "op-code"
• op-code = hex machine code for each ASM instruction

•OPERAND: data/address used by ASM instruction
- function of "addressing mode" (discussed later)
- some instr. don't have operands

$= HEX value
- EX: $12 = 18

•%= BIN value

– EX: %10 = 2

10

10

• none = DEC value

– EX: 12 = 12

10

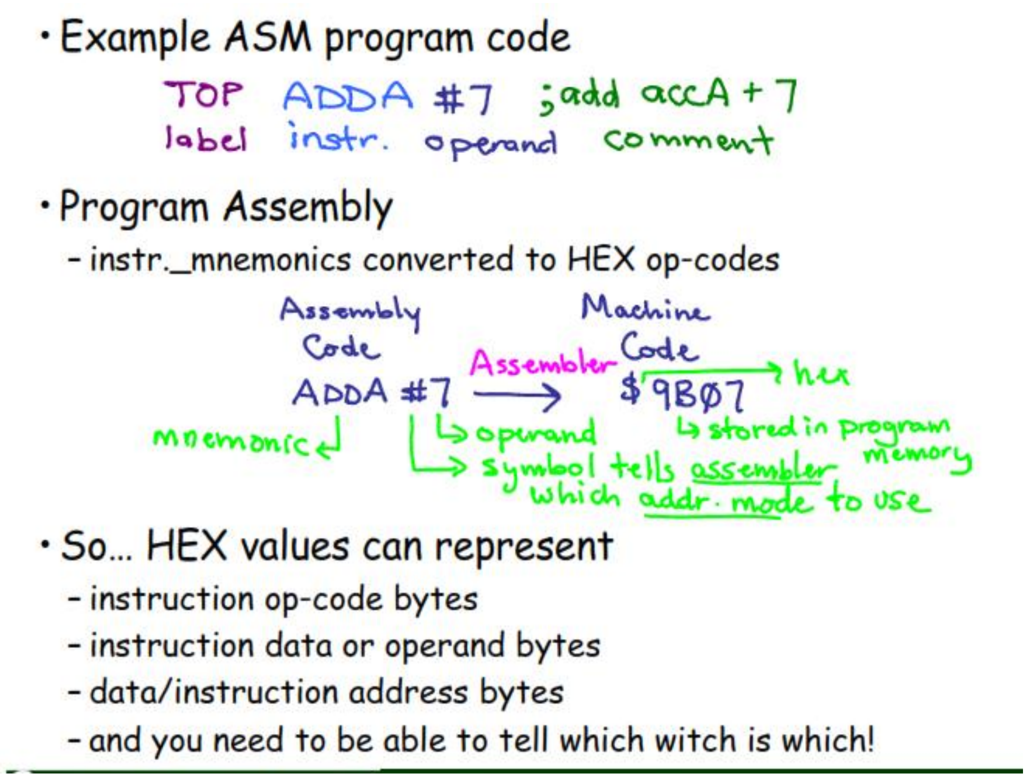
•#denotes a data value (rather than an address)

– EX: ADDA #$A5, adds value $A5 to accA

ADDA $A5, adds value at address $A5 to accA

– illustration to clarify will come soon!

• first need to make things even more complicated ☺



____

Addressing Mode:

An addressing mode specifies how to find/locate the data.

- Inherent
- Immediate

- Extended
- Direct
- Indexed (=Indirect)
- Relative

- Inherent: <u>instruction requires no data</u> form outside the CPU
  - EX: ABA {A ← A+B} or INX {increment iX by 1}
  - inherent instructions have no operands
    - all needed data within CPU registers
- Immediate: <u>data value is in operand</u> (not in memory)
  - easily identified by having a # in the operand
  - EX: LDAA #$F3 {put $F3 into accA} or ANDA #$2C
- Extended: <u>data for instruction is in memory</u>
  - 2-byte operand specifies memory address
  - EX: LDAB $20B4 {put value in $20B4 into accB; B ← <$20B4>}
- Direct: data for instruction is in memory with 1-byte address
  - special case of Extended
    - used for addresses $0000 - $00FF (256 bytes), where MSBy is $00
    - saves program bytes and execution time; eliminates 1 operand value
    - addresses $00-$FF largely used by configuration registers (control I/O functions)
  - EX: ADDA $E3 {A ← A+<$00E3>}

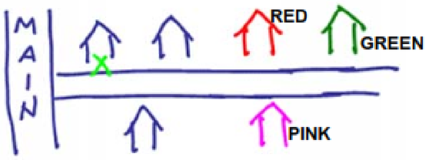
remaining 2 modes are more complicated and will be covered later

<..> notation means the value at this mem. addr.

Relative addressing mode :

"Relative" Concept

- How can we define the address of the red house?



- Absolute Address
  - 3rd House on the left from Main Street

- Relative Address
  - locate by offset from reference
  - references   offsets   offset + reference
    - Green House
    - Pink House
    - 'X'

بيقول ان النوع ده بيستخدم عند عمل

Branches

فقط ،،،و بيبدا يقول تخيل شارع متفرع من شارع رئيسي و فيه مجموعه من البيوت و عايز يعرف مكان البيت اللي لونه احمر ... عنده طريقتين ،،،طريقه مباشره اني اقول هو 3 بيت على يسار الشارع الرئيسي ،،و طريقه غير مباشره "بالنسبه لبيت او مكان مرجعي "

مثلا لو اخدت البيت الاخضر كمرجع هقول ان البيت المطلوب علي يمينه مباشره و اخدت البيت "اكس" يبقى البيت الاحمر على يساره ببيتين ...ده معنى كلمه "ريلاتيف –مرجعي "

مثلا في كود زي ده

```
      LDAA #$10
LOOP  NOP
      DECA
      BNE LOOP
```

بعمل "برانش" على كلمه "لووب" اللي هيا

Label

للسطر الثاني من البرنامج بيقولي و ال اسيمبلر جاي يترجم السطور دي ل اسمبلي بيجي على كلمه "لووب" و بيترجمها كـ

Offset

يعني كأن اخر سطر ده هو البيت "اكس" و السطر الثاني هو البيت "الاحمر "و الـ

Assembler

هو الشخص اللي عايز يروح من البيت "اكس" للبيت الاحمر ...مفروض اقوله هنا يرجع اد ايه عشان يوصل فبيقولي ان فيه قانون عشان تحسب تمشي اد ايه القانون ده هو

- The assembler calculates the offset from current PC value in the following manner:

$$rr = PC_{NEW} - PC_{OLD} - T$$

where:

rr is the relative address(offset),

PC_{NEW} is the destination address

في المثال اللي فوق عنوان الديستنيشن هو عنوان السطر اللي مكتوب فيه لووب,

PC_{OLD} is the address of the branch instruction

في المثال اللي فوق في المربع عنوان ده هو عنوان اخر سطر

T is the number of bytes used by the branch instruction.

T=2 byte

Indexed Addressing:

- It uses one of the two index registers to calculate the address of the data. It uses the index register to refer to the data indirectly

LDAA $04,X

يعني بقوله الريجيستر اللي اسمه "اكس" بيكون فيه عنوان ...خده و ضيف عليه 4 ...هتلاقي العنوان اللي انا عايزك تروح تاخد منه البيانات اللي هتحطها في المخزن "

A (Acc. A).